



Coral8 FAQ (Advanced)

Version 5.4

Copyright © 2004-2008 Coral8, Inc. All rights reserved worldwide.

Worldwide Headquarters: Coral8, Inc.
82 Pioneer Way, Suite 106 Phone: 650-210-3810
Mountain View, CA 94041 Fax: 650-210-3811

Visit us on the web at www.coral8.com or e-mail us at info@coral8.com.

The Programs (which include both the software to which this documentation relates and the documentation itself) are protected by copyright laws and any and all unauthorized reproduction or distribution is strictly prohibited. Patents Pending.

Coral8, Coral8 Engine, Coral8 Server, Coral8 Studio, Coral8 SDK, Continuous Computation Technology, Continuous Computation Language (CCL), Make It Continuous, and CCL Query are trademarks of Coral8, Inc.

Other names may be trademarks of their respective owners.

The Programs contain proprietary information of Coral8, Inc.; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The Programs include third-party proprietary code which is protected by copyright law and may be protected by other intellectual property laws. Required copyright notices and other notices/restrictions related to such third-party code are provided in the text file Studio/doc/Coral8ThirdPartyComponentsUseNotice.pdf, which is distributed with the Programs.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Coral8, Inc. does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Coral8, Inc.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

The Programs are provided with RESTRICTED RIGHTS. Use, duplication or disclosure of the Programs by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights clause in Technical Data and Computer Software clause at DFARS 252.227-7013 or in subparagraphs (c)(1) and (2) of the Commercial Computer Software -- Restricted Rights at 48 CFR 52.227-19, as applicable.

Contractor/Manufacturer is Coral8, Inc. at 82 Pioneer Way, Suite 106, Mountain View, CA 94041 or at (650) 210-3810.

You may not, directly or indirectly, export or re-export: (a) the Programs; or (b) any product (or any part thereof), process or service that is the direct product of the Programs, to any country, person, entity or end user without the appropriate United States and foreign government export licenses. Restricted countries currently include, but are not necessarily limited to, Cuba, Iran, Iraq, Libya, North Korea, Sudan and Syria.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications.

It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Coral8, Inc. disclaims liability for any damages caused by such use of the Programs.

Table of Contents

Preface.....	1
CCL Questions and Answers	3
Questions.....	3
Answers.....	4
Coral8 Studio and Files.....	15
Questions.....	15
Answers.....	15
Index	19

Preface

This document contains answers to more advanced questions than are found in the *Coral8 Frequently Asked Questions (Basic)* document. Specifically, this document contains answers to common questions about the Coral8 Continuous Computation Language (CCL) and the Coral8 Studio.

For an overview of Coral8 Engine, please see:

- *Coral8 Technology Overview*
- *Coral8 Frequently Asked Questions (Basic)*

For Coral8 Server and Coral8 Studio installation and configuration instructions, please see the *Coral8 Administrator's Guide*.

To get started using Coral8, please see the *Coral8 Getting Started* tutorial.

For more help with using the Coral8 Studio interface, please see the *Coral8 Studio Guide*.

For more advanced Coral8 Engine and CCL programming concepts, please see:

- *Coral8 Programmer's Guide*
- *Coral8 CCL Reference*
- *Coral8 Integration Guide*

CCL Questions and Answers

This chapter contains questions and answers about CCL, and especially about the differences between CCL and SQL. For more information about the questions discussed here, please see:

- *Coral8 Programmer's Guide*
- *Coral8 Continuous Computation Language Reference*
- *Coral8 Integration Guide*
- *Coral8 Administrator's Guide*

Questions

[What kinds of windows does CCL include?](#)

[When do continuous joins generate results? What results do they generate?](#)

[Can stand-alone windows be shared by multiple queries?](#)

[I have a query that gets rows from a window and publishes them to a stream. Why do I see rows in the stream that the window policy discarded?](#)

[How do I create an index for faster processing?](#)

[Can I still issue an ad hoc query in CCL, like I can with a database?](#)

[How are timestamps handled? Who inserts them?](#)

[Can the Coral8 Engine be clustered?](#)

[Can I configure additional Coral8 Engines to provide backup in case of failure?](#)

[Does Coral8 provide any security features to restrict access?](#)

[How can I read historical or reference data from a database?](#)

[How can I write streaming data into a database?](#)

[If aggregators can't be used inside a WHERE clause, what do I do?](#)

[What's the difference between the WHERE and HAVING clauses?](#)

[What's the difference between GROUP BY and a window with a PER clause?](#)

[Can I use GROUP BY or PER with multiple columns or expressions?](#)

[Can I join more than two windows, or a stream with multiple windows?](#)

[What's the syntax for an outer join?](#)

[Does CCL support subqueries?](#)

[How do I look at the previous value in the stream?](#)

[Are transactions and commit and rollback commands supported?](#)

Answers

What kinds of windows does CCL include?

CCL includes multiple window types, including:

- A *count-based sliding window* keeps the last N rows, where N is a specified number, for example: `FROM Attacks KEEP 100 ROWS`.
- A *count-based jumping window* accumulates all rows until a specified number of rows is reached, at which point all rows are deleted and the window accumulates rows until the specified number is reached again. For example, `FROM Attacks KEEP EVERY 100 ROWS` resets after each 100 rows.
- A *bucket-based window* is a variation on a count-based window that groups incoming rows based on the value of one or more columns and then keeps a specified number of those groups. For example, `KEEP 3 BUCKETS BY StockSymbol` keeps three buckets of rows based on stock symbol, regardless of how many rows are in each bucket.
- A *time-based sliding window* keeps each incoming row for a specified amount of time. For example, `FROM Attacks KEEP 15 MINUTES` is a sliding window in which the starting and ending times of the specified interval (15 minutes) are continuously advanced as time passes.
- A *time-based jumping window* keeps all rows that come in during a specified period of time, until the time period expires, at which point all rows are deleted and a new period starts. For example the jumping window `FROM Attacks KEEP EVERY 15 MINUTES` "jumps" and resets (starts from 0 rows) every 15 minutes. A variation keeps all rows until a specific time. For example, `KEEP UNTIL '17:00'` resets the window at 5:00 p.m. every day.
- A *sorted window* keeps the top N values on the stream according to the sort order. For example `FROM Attacks KEEP 10 LARGEST ROWS BY Attack.Duration` keeps the ten largest rows received at any given point.
- A *LAST window* keeps the last row received in the stream. This type of window is identical to a rows-based window that keeps one row. For example: `FROM S KEEP LAST PER S.Column` is the same as `FROM S KEEP 1 ROW PER S.Column`.
- An *ALL window* keeps all the rows that ever arrive in the stream, for example, `FROM S KEEP ALL`.
- A *multi-policy window* keeps rows based on both an interval and a row number condition. For example, `FROM Downloads KEEP LARGEST BY FileSize KEEP 10 MINUTES` keeps the row with the maximum file size value within a ten-minute interval.

When do continuous joins generate results? What results do they generate?

Here is the easiest way to think about continuous joins:

Every join has two or more data sources, with at most one of them being a stream without a window, and others being streams with windows. Windows maintain state, but streams without windows do not.

When a row arrives in one of the streams (with or without a window) the row is checked against all other rows held by the other windows in the join. All resulting combinations of rows are checked against the **WHERE** clause. Those that match the **WHERE** clause are published, according to the specifications of the **SELECT** clause. Consider this example:

```
INSERT INTO Result
SELECT *
FROM
S1 AS A,
S2 AS B
KEEP 30 MINUTES
WHERE
A.Id = B.Id;
```

This query has a window on stream S2, but no window on S1. When rows arrive in S1, they are checked against all the rows in the window on S2. The resulting stream in this case has all the columns from A, plus all the columns in B. Note that, when a row B arrives in S2, it is inserted into the window, but the query is not executed. Since there is no state held on S1, B cannot be checked against any A's in S1.

Now let's look at another example:

```
INSERT INTO Result
SELECT *
FROM
S1 AS A KEEP 30 MINUTES,
S2 AS B KEEP 30 MINUTES
WHERE
A.Id = B.Id;
```

In the second example, the state is kept on both S1 and S2. Thus, this query produces output on both rows on S1 and S2, provided, of course, that the incoming rows match the **WHERE** condition.

Finally, it should be obvious why this query would never produce any result and is not allowed:

```
INSERT INTO Result
SELECT *
FROM
S1 AS A,
S2 AS B
```

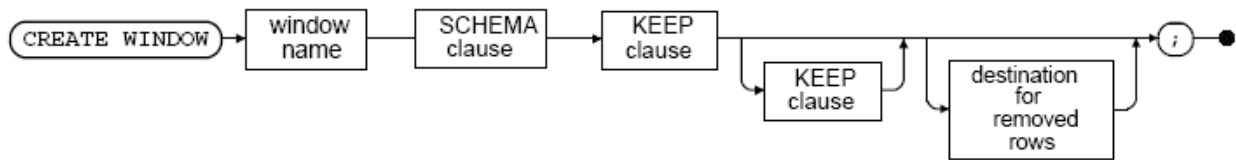
```
WHERE
A.Id = B.Id;
```

No state is held by this join, so neither S1 nor S2 rows would trigger any output. A join query (a query with more than one input source) must have at least one window, and may not have more than one stream without a window.

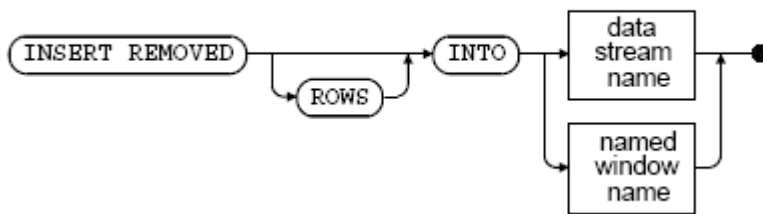
Can stand-alone windows be shared by multiple queries?

Yes, you can achieved this by defining a *named window*. Named windows are similar to materialized views found in relational databases in that they contain data, but they also actively enforce the policy by sorting and removing the data. You create a named window with a CREATE WINDOW statement, which uses the following syntax:

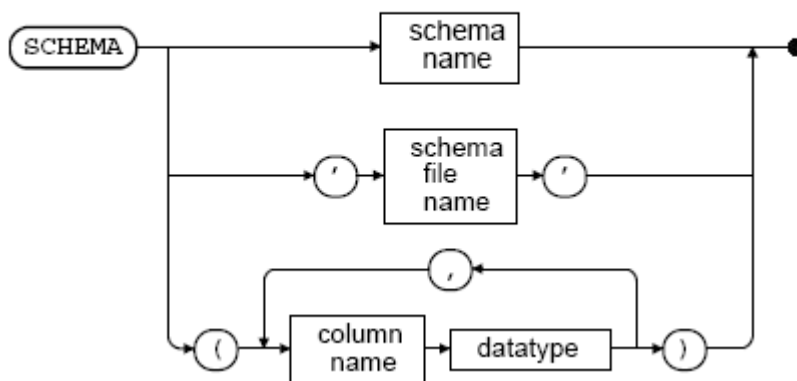
Create-Window-Statement-Syntax:



destination-for-removed-rows:



SCHEMA-Clause-Syntax:



For example:

```
CREATE WINDOW MyWindow SCHEMA 'MySchema.ccs' KEEP 10 MINUTES
```

or:

```
CREATE WINDOW MyWindow SCHEMA (Column1 STRING, Column2 INTEGER)
KEEP 5 ROWS PER Column1;
```

Once a named window is defined, you can publish information into it by using the **INSERT** clause. You can also use the named window as the data source for queries. The ParentChildTracking example, included in your Coral8 Engine installation, demonstrates how named windows may be used.

I have a query that gets rows from a window and publishes them to a stream. Why do I see rows in the stream that the window policy discarded?

Window policies have no effect on which rows are propagated down the query chain, or on how quickly the rows are propagated. Window policies keep information about the state of rows, but do not delay them, or prevent them from traveling down the chain. Window policies also do not prevent expired or discarded rows from propagating down the chain if the rows are not filtered out by other clauses in the query.

How do I create an index for faster processing?

The CCL compiler creates all the needed indexes automatically, so no indexes need to be created manually. The compiler analyzes a query and produces all the optimal data structures, including indexes, that are necessary for the query's continuous processing. These data structures index data by time and by all the keys needed for fast retrieval. For example, consider this query:

```
INSERT INTO Result
SELECT *
FROM
S1 AS A KEEP 30 SECONDS,
S2 AS B
WHERE
A.Id = B.id;
```

CCL automatically detects that stream S1 should be indexed by attribute Id, but that stream S2 does not need to be indexed, as no state for S2 is stored in the query.

Can I still issue an ad hoc query in CCL, like I can with a database?

You can issue ad hoc queries against public windows. Use the **CREATE PUBLIC WINDOW** statement to create a public window:

```
CREATE PUBLIC WINDOW MyPubWindow
SCHEMA Myschema
KEEP 1 HOUR
PROPERTIES
INDEXCOLUMNS = "Symbol, Price";
```

You query the state of a public window with Coral8 SQL, either by clicking **Query Public Windows** from the Coral8 Studio **Debug** menu and then entering your query into a text box, or by using the appropriate method in one of the Coral8 SDKs.

How are timestamps handled? Who inserts them?

It is normally up to the adapter to insert correct timestamps, as required by the application. The Coral8 Engine uses the timestamps provided by the adapter, even if messages arrive out of order (provided the stream is marked as allowing out-of-order messages). Adapters supplied by Coral8 have different ways of providing timestamps. The standard CSV (comma-separated-values) adapter, for example, permits the timestamp to be provided in the incoming data. This adapter can also be requested to add the "current timestamp" to each row sent to the Server. To get the value of the timestamp for a row, use the `GETTIMESTAMP()` function.

Can the Coral8 Engine be clustered?

It is easy to set up multiple versions of the Coral8 Engine to run in parallel, by subscribing to the same stream, or in a pipelined mode, by breaking down processing into stages and running multiple stages on multiple machines. Coral8 Server Manager can deploy a compiled CCL module to multiple Coral8 Servers.

Future versions of Coral8 will support automatic dynamic load balancing across the Server and other advanced clustering features.

Can I configure additional Coral8 Engines to provide backup in case of failure?

You can enable the Coral8 High Availability (HA) feature in an enterprise installation. With HA enabled, if a Coral8 Server stops operating due to a hardware failure or a software crash, another Server replaces the malfunctioning Server, without shutting down the application.

Does Coral8 provide any security features to restrict access?

Yes. You can configure Coral8 Server to restrict specific Server actions and resources to specified IP addresses, host names, users, and/or groups, thus protecting resources from unauthorized use. You set up such restrictions in an access control list. You can also create a user authentication plugin with one of the Coral8 SDKs to verify user identity and group membership.

How can I read historical or reference data from a database?

You can use the Database Subquery feature of CCL. Here a query is issued to the database only when required. The SQL is embedded directly into the CCL query, just like regular SQL subqueries are. The `ParentChildTrackingWithDatabase` example, which is included with your installation of Coral8 Engine, illustrates this method of retrieving data from a database.

How can I write streaming data into a database?

You can write to a database directly from a CCL query by using the CCL Database Statement.

If aggregators can't be used inside a WHERE clause, what do I do?

Just like in SQL, a **WHERE** clause cannot use aggregators. Therefore, the following statement is *not* legal:

```
INSERT INTO Result
SELECT S.Price
FROM
S KEEP 30 SEC
WHERE
S.Price > AVG(S.Price);
-- NOT LEGAL: AGGREGATOR IN WHERE CLAUSE.
```

The simple way around this limitation is to create an intermediate stream that holds both the current and aggregated values, and then to use the value from this stream. For example:

```
INSERT INTO S2
SELECT S.Price as Price, AVG(S) AS AvgPrice
FROM
S KEEP 30 SEC;
INSERT INTO Result
SELECT S.Price
FROM
S2
WHERE
S2.Price > S2.AvgPrice;
```

Another alternative is to use the **HAVING** clause, which does permit the use of aggregators.

What's the difference between the WHERE and HAVING clauses?

WHERE and **HAVING** behave in CCL just like they behave in SQL. Filtering performed by the **WHERE** clause refers to columns in the **FROM** clause and happens before any aggregators are applied. Filtering performed by the **HAVING** clause happens after aggregators are applied and refers to columns and aggregators in the **SELECT** clause. Note that **HAVING** is frequently used with **GROUP BY** to do filtering after **GROUP BY** occurs, but **HAVING** may be used without **GROUP BY**. Please refer to any standard SQL tutorial for more information on the differences between **WHERE** and **HAVING**.

What's the difference between GROUP BY and a window with a PER clause?

GROUP BY is used to define the way data is grouped in queries. Generally speaking, **GROUP BY** affects:

- aggregators (such as **COUNT**, **AVG**, **MIN**, **MAX**, and others)
- window data contents for unnamed windows

- output rows generated by the **OUTPUT EVERY** clause
- output rows generated by the *time-based* **OUTPUT FIRST** clause

A **GROUP BY** clause includes a list of one or more columns. When used in a query with a window and aggregators, the **GROUP BY** causes the aggregator output to generate a row for every unique combination of values in the list.

For example, if the **GROUP BY** lists a column called Times, which contains a set of time value, and another column called Places, which contains names of places, and this **GROUP BY** is used with an **AVG** aggregator, which calculates the average of another column called Temperature, then a row calculating the average temperature is generated for every unique combination of times and places listed in the Times and Places columns. This use of **GROUP BY** is similar to the syntax used in standard SQL. In CCL, the **GROUP BY** clause, used in conjunction with an unnamed window (a window defined in the **FROM** clause) also causes separate window data to be kept for each set of **GROUP BY** values, using the rule specified in the **KEEP** clause for example, `KEEP 10 ROWS`. Here is another example

```
INSERT INTO MaxPrice
SELECT Trades.Symbol, Trades.Volume, MAX(Trades.Price)
FROM Trades
KEEP 10 ROWS
GROUP BY Trades.Symbol;
```

This query keeps ten rows of data in the Trades window for every value in the Symbol column of the Trades stream in the incoming data. `MAX(Trades.Price)` is calculated from these ten rows of data for each symbol. As new rows are added to the window for a given symbol, the oldest rows expire and are taken out of the window, always leaving ten rows in the window for that symbol.

While aggregator calculations are always controlled by the **GROUP BY** clause, the **PER** clause and the **UNGROUPED** keyword in the unnamed window definition may be used to override the grouping of window data contents specified by the list of columns in the **GROUP BY** clause. If you want to establish different groupings for aggregators and window contents, you may use the **PER** clause with the **KEEP** clause to specify a different column (or columns) by which the window contents should be grouped.

For this example, let's first assume that the Trades data stream, used in the previous example, includes a Broker column that lists the broker responsible for the trade. If you wanted to retain ten rows of data for every symbol traded by each broker, you could write this query:

```
INSERT INTO MaxPrice
SELECT Trades.Symbol, Trades.Broker, Trades.Volume,
MAX(Trades.Price)
FROM Trades
KEEP 10 ROWS
GROUP BY Trades.Symbol, Trades.Broker;
```

Note that, in this query, both the window grouping and the aggregator calculation is by Symbol and Broker. If, however, you want to group window data by symbol and broker, but calculate the maximum price based only on the symbol, regardless of the broker associated with it, the query requires a **PER** clause. Now it looks like this:

```
INSERT INTO MaxPrice
SELECT Trades.Symbol, Trades.Volume, MAX(Trades.Price)
FROM Trades
KEEP 10 ROWS PER Trades.Symbol, Trades.Broker
GROUP BY Trades.Symbol;
```

This query specifies that the aggregation should be performed for every symbol, without regard to the broker, but that the window contents should be maintained based on both symbol and broker.

If you want the window data grouping to be eliminated (that is, to have the `KEEP 10 ROWS` rule apply to all rows in the window, regardless of column value) use the **UNGROUPED** keyword in the **KEEP** clause to override the **GROUP BY**, like this:

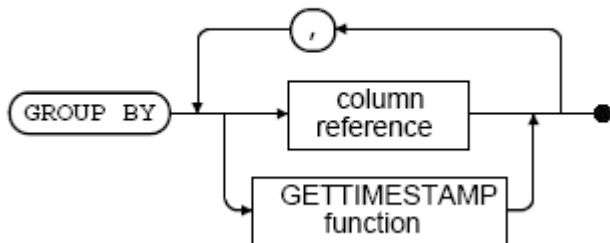
```
INSERT INTO MaxPrice
SELECT Trades.Symbol, Trades.Volume, MAX(Trades.Price)
FROM Trades KEEP 10 ROWS UNGROUPED
GROUP BY Trades.Symbol;
```

In this query, the window only keeps the most current ten rows, regardless of the symbol or broker, but the aggregation continues to be calculated by symbol. Of course, this means that the number of rows used to calculate the maximum price for a given symbol may vary from one (1) to ten (10) at any moment, depending on how many of the ten preceding rows coming into the system are currently held in the window for a given symbol.

Can I use **GROUP BY** or **PER** with multiple columns or expressions?

GROUP BY and **PER** with multiple columns are supported. The syntax for **GROUP BY** is:

GROUP-BY-Clause-Syntax:

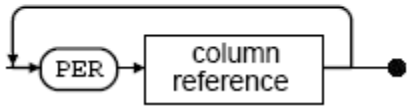


where *column-reference* is one or more column references, for example:

```
GROUP BY Event.Sourceid, Event.Eventid
```

The **PER** clause also supports multiple columns, using the following syntax:

PER-Clause-Syntax:



for example:

```
PER Event.Sourceid PER Event.Eventid
```

GROUP BY and **PER** with an expression will be supported in the future. For now, an intermediate stream may be created, where the expression is saved in a separate column.

Can I join more than two windows, or a stream with multiple windows?

Yes. Coral8 joins may involve more than two data sources, such as multiple windows or a stream and multiple windows. Note that arbitrarily complex windows may be joined as easily as simple windows.

What's the syntax for an outer join?

Coral8 supports left, right, and full outer joins. Here is an example of a left outer join:

```
INSERT INTO Stream3
SELECT *
FROM
Stream1 AS S1
KEEP 100 ROWS
LEFT OUTER JOIN
Stream2 AS S2 KEEP 100 ROWS
ON S1.Ii = S2.Ii
WHERE S1.Ss = S2.Ss;
```

The meaning of an outer join is precisely the same as in traditional relational databases: this left outer join produces results even when there are no row in the second stream matching the join condition `S1.Ii = S2.Ii`. The corresponding column values from the other stream are all `NULL`.

Does CCL support subqueries?

Yes, it does. Currently, three kinds of subqueries are supported:

- **FROM** subqueries
- Scalar non-correlated subqueries
- Database subqueries

Here are examples of the first two types of subqueries. For an example of a database subquery, please see the *Coral8 Frequently Asked Questions (Basic)*.

```

-- scalar subquery --
INSERT INTO ScalarSubOut
SELECT
S1.Price AS Price, S1.Symbol AS Symbol
FROM
StreamIn1 AS S1
WHERE
S1.Price < (SELECT AVG(S2.Price)
            FROM StreamIn2 AS S2 KEEP 3 SECONDS) ;
--
-- same query rewritten as FROM-clause subquery
--
INSERT INTO FromSubOut
SELECT
S1.Symbol AS Symbol1, S1.Price AS Price1,
TO_INTEGER(F2.avg_price) AS Price2
FROM
StreamIn1 AS S1 ,
(SELECT AVG(S2.Price) AS Avg_price
FROM StreamIn2 AS S2 KEEP 3 SECONDS) AS F2
KEEP LAST
WHERE S1.Price < F2.Avg_price ;

```

How do I look at the previous value in the stream?

Use the **PREV()** function. For example, the following query compares a stock's current price with its previous price.

```

INSERT INTO StepUp
SELECT Trades.Symbol,
FROM Trades
KEEP 2 ROWS PER Trades.Symbol
WHERE Trades.Price > PREV(Trades.Price)
GROUP BY Trades.Symbol;

```

PREV can go back more than one row, for example, `PREV(Trades.Price, 5)`, which goes back five messages.

Are transactions and commit and rollback commands supported?

In a traditional database engine, a transaction may operate on zero or more SQL commands, each of which executes to completion and changes zero or more rows; once all the SQL commands in a transaction have completed, the entire transaction may be committed or rolled back.

In Coral8 Engine, however, CCL commands do not execute once and then finish; instead, each query runs continuously, operating on each new row that arrives. Since CCL statements never "finish" executing, the Coral8 system does not support the concept of finishing statements and

Coral8 FAQ (Advanced)

then committing or rolling them back. CCL does not have transactions, or commit or rollback statements.

Note, however, that some transactional properties, such as durability, are supported if state persistence is enabled for the modules of a given project. Even if a machine crashes, the state persists in local storage and is recovered when the system becomes available.

Further, the Guaranteed Delivery feature of Coral8 Engine, in conjunction with persistence, allows you to specify that rows in a stream are delivered exactly once, with none being duplicated or dropped. For a complete guaranteed processing system, you must also use adapters with Guaranteed Delivery enabled and configure High Availability.

Coral8 Studio and Files

This chapter contains questions and answers about Coral8 Studio and about data organization within Coral8 applications. For more information, please see the *Coral8 Studio Guide*.

Questions

[What is Coral8 Studio?](#)

[What is a Coral8 Project?](#)

[What is a CCL Module?](#)

[Can modules be parameterized and reused?](#)

[What is a Workspace?](#)

[What is an Environment?](#)

[How can I view streams to debug my application?](#)

[How do I know whether a project is running, if I don't open a stream viewer?](#)

[What kinds of files does Coral8 use and where are they located?](#)

Answers

What is Coral8 Studio?

Coral8 Studio is a User Interface application that lets users create streams and stream schemas, develop CCL queries, organize them into query modules and projects, compile and load projects into workspaces, load adapters, view streams at run-time, and so on.

What is a Coral8 Project?

Projects contain reusable groups of CCL queries (query modules), combined with one or more data streams and adapters. Every project includes at least one query module, which may contain one or more submodules. Projects must be loaded into a workspace for editing and execution, and, once loaded, appear under one of the workspaces in the Coral8 Studio. Project definition information is stored in a file with the extension `.ccp`, while query module code is stored in a file with the extension `.ccl`. After being compiled, projects are translated into a low-level language that can be executed by the Coral8 Server. The resulting `.ccx` files are not visible in Studio, but are executed when a project is started.

Note that, once they are started, projects keep running until they are either explicitly stopped, or the Coral8 Server on which they are running is shut down. Projects do not stop running if you simply disconnect them from the workspace or close Studio.

What is a CCL Module?

In its simplest form, a CCL query module is just a collection of CCL queries. CCL modules make it easy to organize queries into logical units, and also to organize a large application into easily manageable parts. Every Coral8 project contains at least one main query module, which may contain additional submodules. Query module code is stored in a file with the extension `.ccl`.

Can modules be parameterized and reused?

Yes, they can. A module may optionally be parameterized with any of the following:

- *Input Streams*: formal names to be used for module input streams
- *Output Streams*: formal names to be used for module output streams
- *Parameters*: scalar parameters used at compile time

For example, suppose we develop a module that takes in a stream from an RFID reader, and counts the number of readings per unit of time. The module could look like this:

```
INSERT INTO RFIDCounts
SELECT Count(*)
FROM RFIDRaw KEEP 10 SEC
OUTPUT EVERY 10 SEC;
```

This module hard-codes the names of the streams, as well as the parameter `10 SEC`. But what if somebody wants to use this module with other input and output streams, and with a different interval? In this case, it is possible to define an interface to this module, consisting of formal input streams `InRFIDRaw` and `OutRFIDCounts`, and a parameter `Duration`. Then the query can be rewritten as:

```
INSERT INTO OutRFIDCounts
SELECT Count(*)
FROM InRFIDRaw KEEP $Duration SEC
OUTPUT EVERY $Duration SEC;
```

Please note that:

- Formal input and output stream names do not use the dollar sign character (\$) but scalars do.
- Parameters are specified in the Parameters tab for the module.
- When a project containing the module is loaded, the connections for the stream and values for scalar parameters are specified in the Bindings tab.
- The notion of formal names and connections is modeled on how parameters are passed to functions in any high-level language like C++ or Java. Each function declares formal parameters and uses them. When the function is called, the caller binds actual values (in this case called connections) to the formal values.

CCL goes far beyond SQL in its concepts of modules and parameterized modules. Unlike CCL, SQL does not let you create a set of queries and then apply them to different tables.

What is a Workspace?

Workspaces are used to isolate different users or projects from one another within the Coral8 Engine. You and/or your system administrator decide exactly how many workspaces to use and how to use these. For example, you may decide that each user should get their own workspace, and that there should be a shared test workspace, a production workspace, and so on. By default, each Coral8 environment comes with one enabled workspace. This workspace is called Default. The Coral8 Studio allows you to create and destroy workspaces, and to connect or disconnect from workspaces. You may connect to a single workspace, or to multiple workspaces, running on the same or different Servers. For more information about working with workspaces in Studio, please see the *Coral8 Studio Guide*.




What is an Environment?

An environment contains the list of workspaces to which Studio users are connected. The environment configuration is stored in a `.cce` file. An environment is strictly a Studio feature, and does not exist on the Server. If you close Studio and then reopen it, it opens to the same environment, containing the same workspaces as it did before you last closed Studio.

How can I view streams to debug my application?

The Coral8 Studio allows you to view every stream in the system, including streams connected to main modules, or their submodules, input and output streams and so on. For instructions on viewing a stream, please see the *Coral8 Studio Guide*. A debugging feature is also available in the stream viewer. This feature allows you to "step through" the rows in a stream and determine what other rows in other streams arrived at the same time, or shortly before or after, the selected row. Debugging features are also discussed in the *Coral8 Studio Guide*.

How do I know whether a project is running, if I don't open a stream viewer?

You can tell the status of a project (which contains your query, or a submodule with your query) by looking at its icon. When the project is not running, its icon looks like this: . When the project is running with no problems, the icon looks like this: . If the project encounters a problem, its icon looks like this: . If a project is not running properly, select it, and look at the Studio Status View and Output View for information about its status. The Status View shows the project's status, while the Output View lists any errors encountered by the project.

What kinds of files does Coral8 Engine use and where are they located?

Coral8 uses several kinds of files:

Coral8 FAQ (Advanced)

- `.ccp` files contain information about the Coral8 project, including the parameter and stream information of the project's main module. `.ccp` files are often called project files, and, when loaded, appear in the Coral8 Studio as projects.
- `.cc1` files are XML files, which contain CCL statements (queries). Every query module is contained in its own `.cc1` file. These files also store parameter and input/output stream information associated with the current module's submodules.
- `.ccs` files are Studio-created XML files that contain schema definitions. These schemas define the column names and datatypes of data streams, windows, and database subqueries.
- Sample data files are contained in `.csv` (comma separated values) files. The `.csv` files that come with your Coral8 Engine are a part of the examples included with every Coral8 installation. These files must reside in the same location as the Coral8 Server, in order for examples to run properly. You may also find it useful to create your own sample data files during the application development and testing phases.

`.ccp`, `.cc1` and `.ccs` files reside on the same machine as your Coral8 Studio installation. These files may be checked into a source control system, if desired. The files are compiled by Coral8 Studio and sent to the Server automatically when you run a project.

Coral8 also uses two other kinds of files, although you do not need to modify these files in any way under normal circumstances.

- `.cce` files contain Studio configuration information, such as the list of open workspaces.
- `.ccx` files contain compiled `.cc1` files. `.ccx` files are produced by the CCL compiler and are sent by Studio to the Server.

Index

A

aggregator 3
AVG 3

B

Bindings 15

C

CCL 3, 15
ccp 15
ccs 15
Column 3
Coral8 Engine 3
Coral8 SDKs 3
COUNT 3
CREATE PUBLIC WINDOW 3
CREATE WINDOW 3
CSV 3

D

Database Subquery 3

F

FROM 3

G

GETTIMESTAMP 3
GROUP BY 3
Guaranteed Delivery 3

H

HAVING 3
High Availability 3
HOUR 3

I

Input Streams 15
INSERT 3

J

Java 15
jumping window 3

K

KEEP 3
KEEP ALL 3
KEEP LAST 3
KEEP LAST PER 3
KEEP UNTIL 3

L

LEFT OUTER JOIN 3

M

MAX 3
MIN 3

O

OUTPUT EVERY 3, 15
OUTPUT FIRST 3
Output Streams 15

P

Parameters 15
PER 3
PREV 3

S

SECONDS 3
SELECT 3
SQL 3, 15

Coral8 FAQ (Advanced)

submodule 15
subqueries 15
W
WHERE 3

Workspace..... 15
X
XML..... 15